

# Requirements and Analysis Document

Viktor Franzen, Tobias Lindroth, Spondon Siddiqui, Alexander Solberg

October 28, 2018

## 1 Introduction

The purpose of the project is to design and create a messaging application in which multiple users can communicate with another. To differentiate from other messaging applications, the main purpose is that multiple users can join a group to communicate primarily on desktops or laptops. Students working in groups will benefit from the application as it provides a simple, private space for them to communicate and share information. As group work is common within education, the application serves a purpose for higher education and is therefore required to simplify communication between groups.

The application will be a cross-platform desktop application with a graphical user interface. The aim is develop the application so it is easy and pleasant to use, as well as have the possibility of being extended into a more complex messaging application. First time users get the possibility to enter the application, identify themselves, and thereafter search for groups they wish to join and communicate with. Once active, the user can view, send, and receive messages.

### 1.1 Definitions, acronyms, and abbreviations

GUI: Graphical User Interface; How the application looks.

User Story: An informal description of a feature of a software system, written in natural language. Often written from the perspective of an end user.

MVC: Model-View-Controller; A design pattern that separates the logic from the view which makes it easy to reuse the model or change view.

UML: Unified Modeling Language; A general-purpose, modeling language used in the development of object oriented projects. It provides a standard way for the design of a system to be visualized.

Channel: A chat group; A group where users of the application can join and send messages. Other users in the same group can see those messages.

JSON: JavaScript Object Notation; A language independent data format.

## 2 Requirements

### 2.1 User Stories

**Story Identifier: STR00 Story Name: Send message**

**Description**

As a user I want to be able to send a message to a channel so that other users in the channel can see it.

**Acceptance criteria**

Functional

- A user can send a message to a channel
- A user can't send an empty message to a channel
- There is a field where the user can write messages
- There is a send button
- All group members receive messages from the channel

Non-functional

- Security - Are users that are not members of a channel prevented from seeing the channel's messages?
- Documentation - Is the code well documented?
- Testability - Can the code be tested in some way?

---

**Story Identifier: STR03 Story Name: Create public group**

**Description**

As a user I want to be able to create a public group conversation so that multiple people can talk together

**Acceptance criteria**

Functional

- There is a button the user can press to create a new group
- There is a field where the user can give the group a name
- There is a field where the user can give the group a description

- Other users can join the group

Non-functional

- Response time - Can other users join the channel directly after it is created?
- Documentation - Is the code well documented?
- Testability - Can the code be tested in some way?

---

**Story Identifier: STR02 Story Name: Use Identification**

**Description**

As a user I want to be able to use identification so the other users know who I am.

**Acceptance criteria**

Functional

- I can choose an identification
- Other people can see my identification

Non-functional

- Security - Two users can't have the same identification
- Documentation - Is the code well documented?
- Testability - Can the code be tested in some way?

---

**Story Identifier: STR01 Story Name: Git repository**

**Description**

As a team we want everyone to have the same Git repository so we can work on the same project without conflicts

**Acceptance criteria**

Functional

- There is a Git repository online
- Each group member has access to the repository
- Each group member understands how to use Git

Non-functional

- ...

---

**Story Identifier: STR09 Story Name: Overview of channels**

**Description**

As a user I constantly want to to see an overview of all my conversations so i can keep track of the channels I'm active in.

**Acceptance criteria**

Functional

- There is a list with the channels the user is a member in
- The list is updated when joining a channel
- The list is update when leaving a channel

Non-functional

- Documentation - Is the code well documented?
- Testability - Can the code be tested in some way?

---

**Story Identifier: STR11 Story Name: Find channels**

**Description**

As a user I want to be able to see public channels with a specific topic so i can join a channel I'm interested in.

**Acceptance criteria**

Functional

- There is a field where you can search for channels
- There is a join button for channels in the search result
- You become a member of the channel when the join button is pressed

Non-functional

- Response time - The search process is fast and the results are displayed immediately.
- Documentation - Is the code well documented?
- Testability - Can the code be tested in some way?

---

**Story Identifier: STR18 Story Name: Save Channel Data**

**Description**

As a user I want my data and messages to be saved somewhere so it exists if I need it again

### **Acceptance criteria**

#### Functional

- The user's account is saved
- The user can log into an existing account
- The users data remains unchanged after login
- The user only sees their own data
- The user doesn't actively have to save the data

#### Non-functional

- Security - Can the password be matched without revealing the data?
- Capacity - How much data can the system store and for how long?
- Reliability - Is the data transferred in a reliable way?
- Documentation - Is the code well documented?
- Testability - Can the code be tested in some way?

---

**Story Identifier: STR08 Story Name: See old messages**

### **Description**

As a user I want to be able to look back at a conversation so I can view information I've forgotten

### **Acceptance criteria**

#### Functional

- The messages are saved
- The user can view the messages
- The messages return to the correct conversation
- The messages remain in the correct order
- The user doesn't have to actively save the messages

#### Non-functional

- Reliability - Is the data transferred in a reliable way?
- Security - Are users that are not members of a channel prevented from seeing the channel's messages?
- Documentation - Is the code well documented?

- Testability - Can the code be tested in some way?
- 

**Story Identifier: STR04 Story Name: Leave channel**

**Description**

As a channel member I want to be able to leave a channel conversation if needed so that I don't receive unwanted messages

**Acceptance criteria**

Functional

- There is a button the user can press to leave a channel
- Other users in the channel get an indication that a user has left

Non-functional

- Security - The user who left can no longer see that channel's messages
  - Documentation - Is the code well documented?
  - Testability - Can the code be tested in some way?
- 

**Story Identifier: STR05 Story Name: Kick channelmember**

**Description**

As an administrator I want to be able to kick people from a channel if needed so that unwanted people cannot see my private conversations.

**Acceptance criteria**

Functional

- There is a list with all members of the group
- The administrator can choose a member and press a button to kick the chosen member
- The member who got kicked gets an indication that he/she has been kicked
- The former member can't see the groups messages anymore

Non-functional

- Response time - The search process is fast and the results are displayed immediately.
- Documentation - Is the code well documented?
- Testability - Can the code be tested in some way?

---

**Story Identifier: STR07 Story Name: New message identification**

### **Description**

As a user I want an indication that I've received a new message so that I don't miss any messages.

### **Acceptance criteria**

Functional

- Groups with non-read messages are "highlighted" in some way
- The marking disappears when the user has read the message

Non-functional

- Documentation - Is the code well documented?
- Testability - Can the code be tested in some way?

## **2.2 User interface**

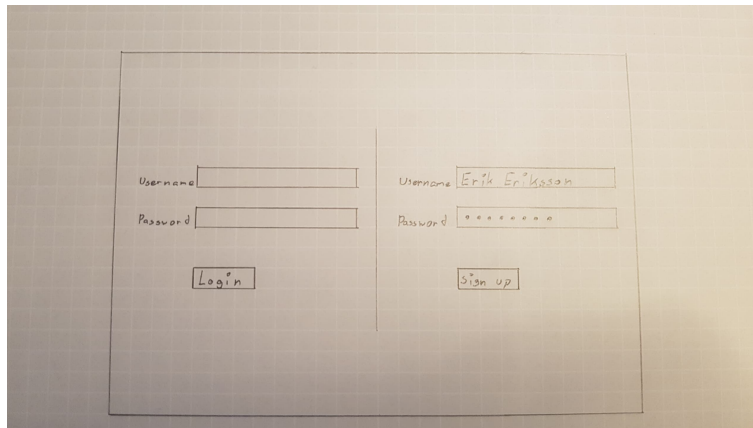


Figure 1: The log-in screen version 1

When the application is started the user is met with a view for logging in or signing up. If the user has an existing account, they can use the fields to the left to log in to it. If they wish to create a new account, they can do so using the fields to the right. The user has to type in a username and a password for either of these to work.

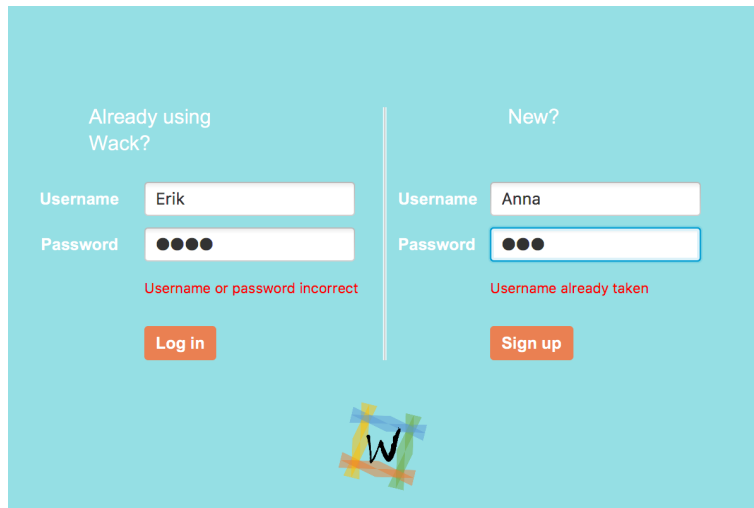


Figure 2: The log-in screen version 2

This is the second iteration of the login view. Color has been added to the background, buttons and text. The view now displays an error message if the user has inputted an incorrect username or password, or if they try to create a user with a name that is already taken.

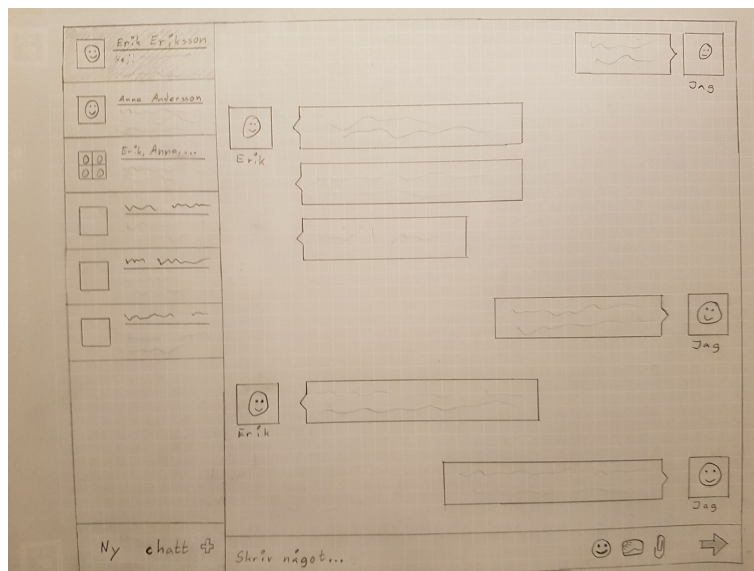


Figure 3: The main view version 1



This is the first iteration of the main view, which will show after the user has logged in or signed up. Here, they can view the channels they are in to the left, and the channel they are currently in is shown to the right/center of the screen. Here, the history of the channel's messages are shown, and the messages are displayed next to the profile picture of the user who sent them. The field at the bottom of the screen is used for typing in messages, and the arrow-button in the bottom-right corner is used for sending messages. The button in the bottom-left corner is used for creating a new channel.

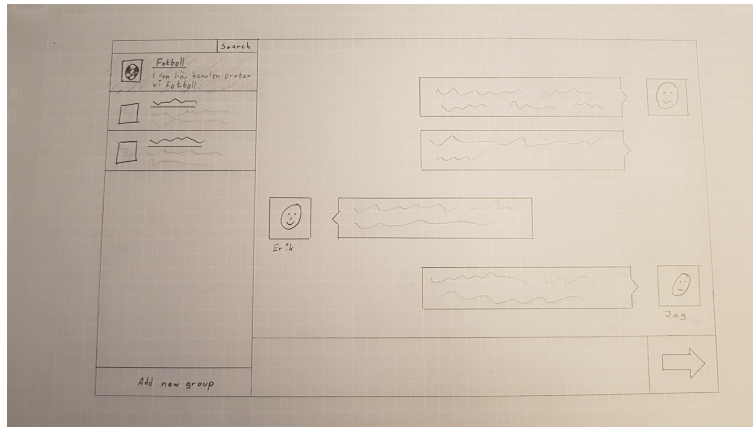


Figure 4: The main view version 2

This is the second iteration of the main view. The application has been made wider and less squarish and the channels are now displayed differently. Now, the name and the description of the channel is shown, as opposed to the users and the latest messages of the channel. Furthermore, a searchbar has been added to the top-left corner to allow users to search for a desired channel.

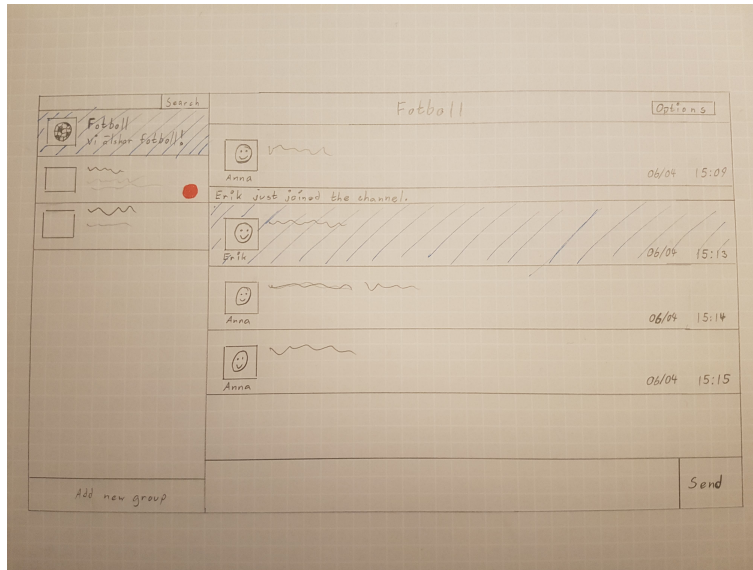


Figure 5: The main view version 3

This is the third iteration of the main view. The way the messages are shown has been simplified to make it more intuitive for the user. The messages now more clearly display who has said what. In addition, a light blue color has been added to signify which channel is selected, and which messages the user has written.

The channel now notifies channel members when a user has joined or left the channel. These messages are disparate from regular user-written messages in appearance; they are thinner and don't have a user linked to them. This makes it easier to differentiate between notifications and messages. The messages now also display the time and date of their sending.

A banner has been added to the top with the name of the current channel. Furthermore, a red dot is now displayed whenever the user has unread messages in another channel, so the user can keep themselves up to date with all of their channels. Lastly, an options button has been added to the top-right corner of the application, which lets the user leave the channel.

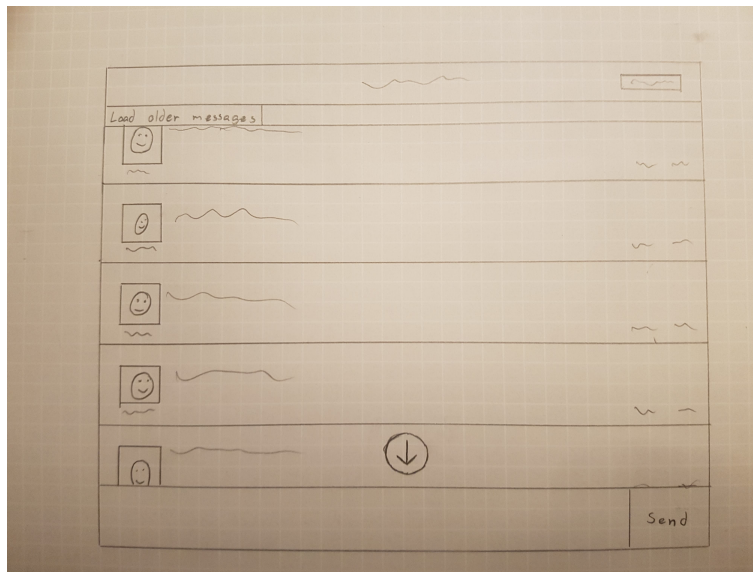


Figure 6: Messages from a single channel

Two new functions has been added to the application. One is a button at the top of the view, just beneath the banner, when the user has scrolled up to the oldest loaded message. This button let's the user load older messages, if they should choose to do so. This is because the application does not load in all the messages in a channel at once, out of convenience for the user; more often than not, the user does not wish to see messages from long ago. However, if they would want to, the possibility is there.

Additionally, there is now a circular button with an arrow at the bottom of the application. This button is shown whenever the user has unseen messages in the current channel, perhaps because they are viewing older messages. This button puts the user at the bottom of the channels messages, as the application does not scroll down automatically.

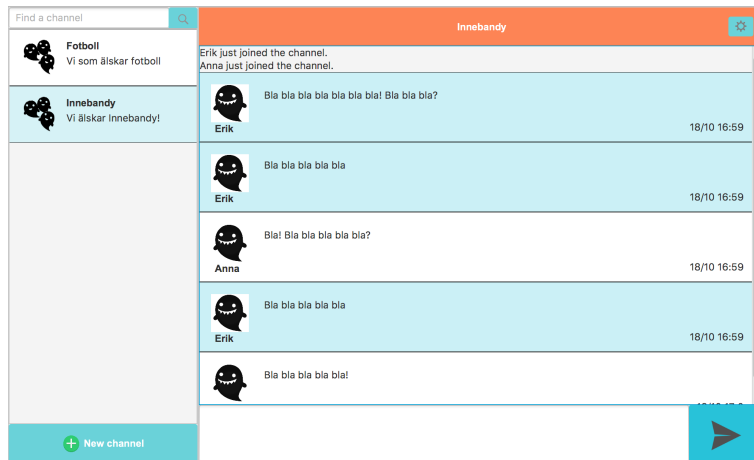


Figure 7: The main view version 4

This is the fourth iteration of the main view. A color scheme has been added to the entire application; the colors light blue and orange, which are complementary and therefore fit well together. The buttons now use symbols instead of text and stick out thanks to their color. The options button now include a list of the channels members, and if a user is the administrator of the group, a button to kick unwanted members.

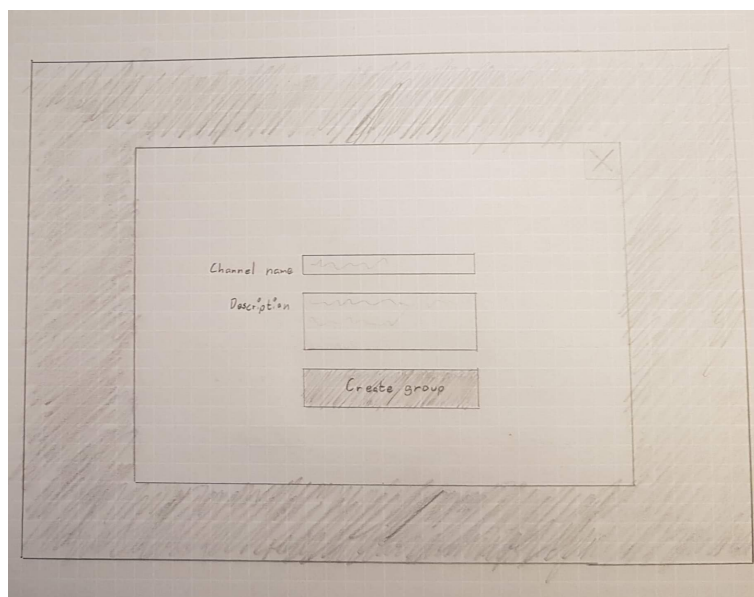


Figure 8: Creating a new channel

This is the view for creating new channels. The view is shown when the user presses the "Add new group" button at the bottom-left of the main view, and is shown on top of the main view. Moreover the main view is darkened to highlight the creation-view. In order to create a new channel, the user has to type in a name and a description for the channel. The user can choose to cancel the creation by either clicking on the x-button in the top-right corner, or by clicking outside on the view, on the main view behind it. The user is then brought back to the main view.

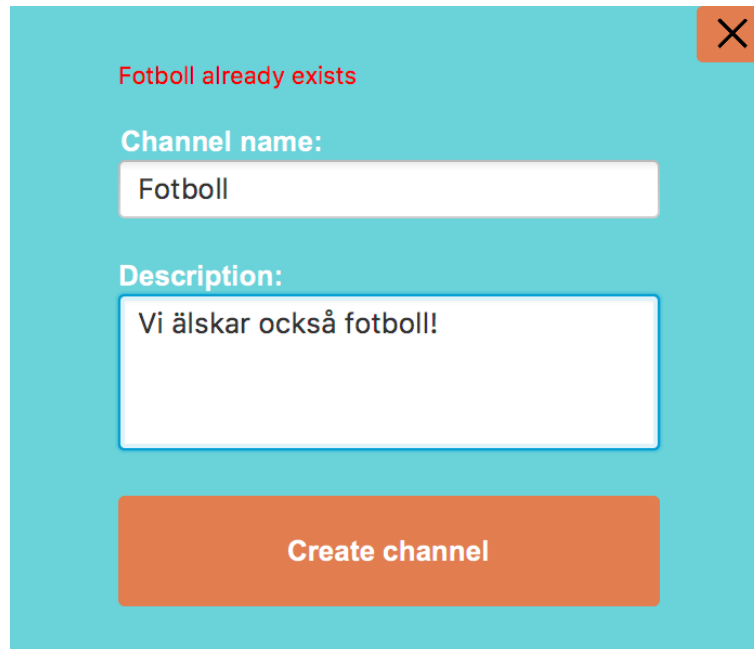


Figure 9: Creating a new channel version 2

The box for creating new channels has been colored according to the color scheme and the entire box has been made smaller. The view now also displays an error message if the user tries to create a channel with a name that already exists.

### 3 Domain model

The domain model consists of five objects. Server, channel, user, client and message. Together, they form the core of the application.

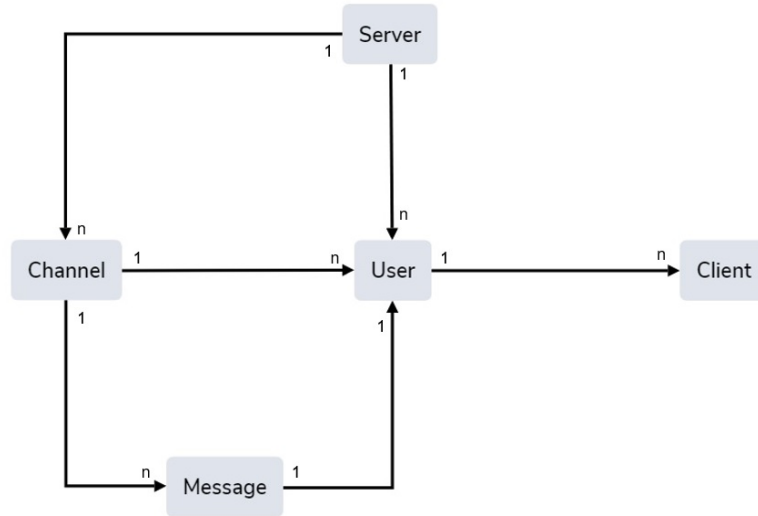


Figure 10: Domain model

### 3.1 Class responsibilities

The server class is responsible for keeping track of all the existing users and channels.

The channel class holds all the messages written to it and keeps track of all the members. You can use the channel to send a message, and then the channel is responsible for delivering this message to all the members of the channel.

The user class is responsible of keeping track of its clients and forwarding all the received messages to the clients.

The client is responsible for being observable and forwarding any messages it receives to any eventual listener.

The message class is responsible for all the information regarding a message. The message content, who sent it, and when the message was created.

## 4 References